TAMPEREEN
AMMATTIKORKEAKOULU

JOHDATUS DEVOPSIIN

Case blogin REST API

Anton Lehmus

Opinnäytetyö
Toukokuu 2018
Tietotekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikka
Ohjelmistotekniikka

LEHMUS ANTON:
Johdatus DevOpsiin
Case blogin REST API

Opinnäytetyö **26** sivua, joista liitteitä 0 sivua.
Toukokuu 2018

Opinnäytetyön tarkoituksena oli tarjota johdatus DevOps käytäntöön esimerkki ohjelmiston avulla. Ohjelmisto oli yksinkertainen REST tyyppinen henkilökohtaisen blogin rajapinta Node.js:llä toteutettuna. Ohjelmiston tavoitteena oli tarjota palvelinpuolen toiminnallisuus Docker-konttiin pakattuna.

DevOps on sanojen development ja operations sanojen yhdistelmä. DevOps käytännössä yhdistyy ohjelmistokehitys, -ylläpito ja laadunvalvonta käytännöt runsaan automaation avulla. Docker on suosittu muun moassa koska se helpottaa automaattista julkaisua.

Ohjelmisto suunniteltiin tukemaan vain yhtä pääkäyttäjää joka voi muokata ja tuottaa sisältöä. Blogissa oli kolme tietorakennetta; postaus, kappale ja tagi. Sovelluksen kehityksessä pyrittiin käyttämään mahdollisimman paljon kolmannen osapuolen kirjastoja kehityksen nopeuttamiseksi.

Ohjelmiston kontitus toteutetiin Docker Composella. Docker Compose on työkalu, joka mahdollistaa useiden konttien määrittelyn ja hallinnan yhden tiedoston avulla. Tietokantana käytettiin PostgreSQL ja sen käsittelyyn Objection.js ORM:ia. Rajapinta toteutettiin Express ohjelmistokehyksen päälle.

Ohjelmiston kehityksessä oli pienistä konfiguraatio virheistä johtuvia ongelmia kontituksen kanssa. Ohjelma ei päässyt sille asetettuihin tavoitteisiin, osin aikataulu rajoitteista johtuen. Testiautomaatio jäi uupumaan täysin, joten ohjelma ei ole kelvollinen esimerkki DevOps tyyppisestä projektista. Sovellus on kuitenkin kontissa toimiva ja se pystyy toimimaan yksinkertaisen blogin palvelinpuolena.

Asiasanat: DevOps, Node, Docker,Express, REST

ABSTRACT
Tampere University of Applied Sciences
Degree programme in ICT Engineering
Software Engineering

LEHMUS ANTON:
Introduction to DevOps
Case blogs REST API

Bachelor's thesis **26** pages, appendices 0 pages
May 2018

The purpose of this bachelor's thesis was to offer an introduction to the DevOps practice with a help of a small piece of example software. The example software was a RESTful back end for a personal blog developed with Node.js. The goal of the software was to be minimal yet functional back end packaged in a Docker container.

DevOps is a clipped compound word of development and operations. DevOps aims to unify software development and software operations and automate as much work as possible. Docker is popular in DevOps practice because it offers tools for easy automatic deployment.

The software was designed to support one main user that has the right to modify and create content. The blog consists of posts, paragraphs and tags. The goal was to utilize third party libraries to speed up the development process as much as possible.

The software was containerized with Docker Compose. Docker Compose is a tool that allows multiple containers to be controlled with a single command and to be configured in a single file. PostgreSQL was chosen to be the database and Objection.js the ORM. The API was developed with Express.

There were some problems with containers caused by minor errors in configuration files. The software did not hit the goals set for it party because of time constraints. The software does not have any kind of test automation so it is not a suitable example of a DevOps type of project. The software was successfully containerized though and it offers the functionality that was planned for it. Even though the code and its documentation is not that great, the API has good usage documentation.

Key words: DevOps, Node, Docker,Express, REST

SISÄLLYS

LYHENTEET JA TERMIT

REST	”REpresentational State Transfer” on arkkitehtuurimalli HTTP-ohjelmointirajapintojen toteuttamiseen.
API	”Application programming interface” eli ohjelmointirajapinta on rajapinta, jonka avulla ohjelmat voivat vaihtaa tietoa keskenään.
Node.js	Alustariippumaton JavaScript ajoympäristö, joka perustuu Googlen V8 JavaScript-moottoriin.
Express.js	Node.js web-ohjelmistokehys (framework)
ORM	“Object-relational mapping” mahdollistaa tietokannan käyttämisen olio-ohjelmonti paradigmalla
JWT	”JSON Web Token” on JSON pohjainen standardi käyttöoikeustietueiden (access token) toteuttamiseen
JSON	JavaScript Object Notation on täysin tekstipohjainen ohjelmointikieli-riippumaton datan esitystapa.
Sovelluskehys	framework, kokoelma ohjelmistoja ja rajapintoja
Docker	Käyttöjärjestelmätason virtualisointia tarjoava ohjelmisto
Kontti (container)	Docker ”virtuaalikone”
CRUD	Create, read, update and delete
Väliohjelmisto	(Middleware) on ohjelmisto, joka mahdollistaa eri ohjelmistojen tai ohjelmiston osien toiminnan.
URL	(Uniform Resource Locator) Resurssin sijaintia osoittava merkkijono, esimerkiksi web-osoite.

1 JOHDANTO

Opinnäytetyön tarkoituksena oli tarjota johdatus DevOps käytäntöön esimerkki sovelluksen avulla. Sovelluksen tavoitteena on olla yksinkertainen ja helposti jatkokehitettävä henkilökohtaisen blogin palvelinpuoli JavaScriptillä. Sovelluksen tarkoitus on tarkoitus tarjota kaikki yksinkertaiseen blogiin liittyvä tietojen käsittely Docker-konttiin pakattuna.

Sovellus jakautuu kolmeen osaan: ohjelmistoon, tietokantaan ja virtualisointiin. Tietokanta on vain tietovarasto eikä sitä käsitellä tietorakenteita enempää. Virtualisointi toteutetaan Dockerin avulla. Ohjelmisto tulee olemaan REST tyyppinen rajapinta Express.js ohjelmistokehyksen päälle toteutettuna. Ohjelmiston ei siis ole tarkoitus olla kokonainen blogialusta vaan pelkkä palvelinpuoli, käyttöliittymä tullaan toteuttamaan toisessa projektissa.

Rajapinnan tulee olla mahdollisimman yksinkertainen, mutta tarjota kuitenkin julkaisuvalmiin blogin toiminnallisuus. Virtualisoinnin tavoite on mahdollistaa DevOps tyyppinen ohjelman kehitys ja jakelu. Tavoitteena ei ole luoda malliesimerkkiä DevOps projektista vaan tarjota minimivaatimukset täyttävä esimerkkiprojekti.

Esimerkkisovelluksen koodi julkaistaan avoimen lähdekoodin lisenssillä GitHubissa https://github.com/AntonLehmus/my_blog

2 TYÖKALUT

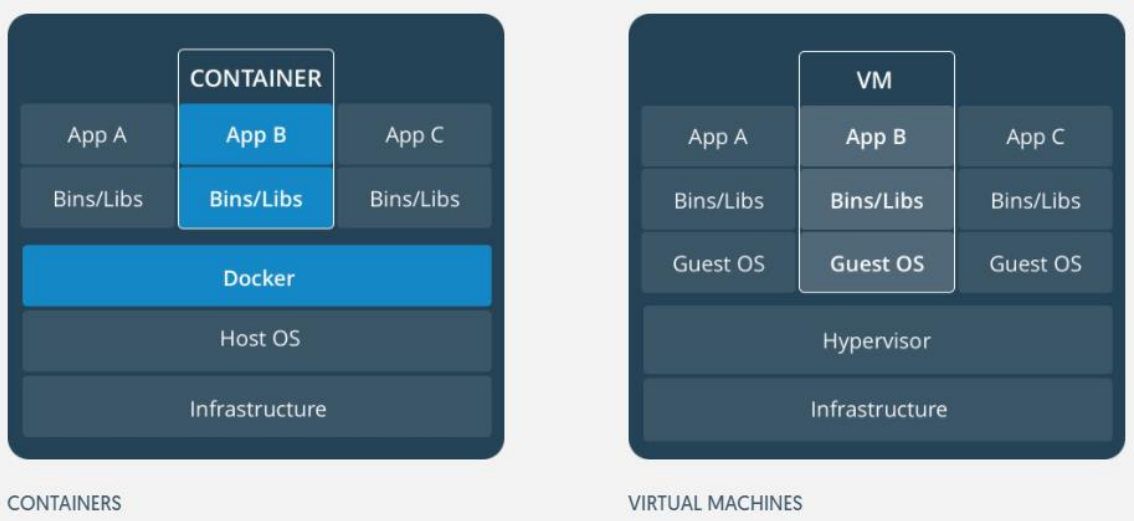
2.1 REST

REST eli Representational State Transfer on erittäin suosittu arkkitehtuurimalli HTTP-rajapintojen toteuttamiseen. Malli asettaa seuraavat arkkitehtuurilliset määritelmät: yhtenäinen rajapinta, asiakas – palvelin arkkitehtuuri, tilattomuus, välimuistetettavuus (eng.cacheable) ja kerrostettu järjestelmä. Yhtenäinen rajapinta tarkoittaa sitä, että jokaiselle resurssille on yksi looginen URL ja että kaikki data esitettään samassa formaatissa ja resurssien käsittelyyn käytetään loogisia HTTP metodeja, esimerkiksi GETiä hakemiseen ja DELETEä poistamisen. Asiakas-palvelin tarkoitta yksinkertaisesti sitä, että ohjelmistokokonaisuus on jaettu toisistaan
riippumattomiin asiakas- ja palvelinohjelmistoihin. Tilattomuus tarkoittaa sitä, että jokainen pyyntö käsitellään samalla tavalla riippumatta asiakasohjelmiston tilasta. Välimuistetettavuus tarkoittaa sitä, että jokainen vastaus ilmoitta kuinka pitkäksi aikaa sen tiedot voidaan jättää välimuistiin ja olettaa muuttumattomaksi. Kerrostettu järjestelmä tarkoittaa järjestelmän jakamista pieniin osiin, joista jokaisella on vai yksi tehtävä.

RESTin hyötyjä ovat suorituskyky, skaalautuvuus, yksinkertaisuus, muokattavuus, luotettavuus.

2.2 Docker

Docker on virtualisointityökalu, joka tarjoaa jaetun käyttöjärjestelmäytimen (shared kernel space) ja eristetyn ajoympäristön (isolated user space). Docker virtuaaliympäristöä kutsutaan kontiksi (container). Docker kontit ovat paljon perinteisiä virtuaalikoneita pienempiä ja nopeampia sammumaan ja käynnistymään. Perinteiset virtualisointiratkaisut virtualisoivat koko käyttöjärjestelmän (KUVA 1). Dockerille on saatavilla kattavasti valmiita kontteja. Konteille voidaan asetta ympäristömuuttujia ja antaa kokonaisia konfiguraatiotiedostoja esimerkiksi web-palvelinta varten. Ohjelmistot voidaan joko asentaa suoraan konttiin, jolloin luodaan uusi konttikuva (eng. image), tai siirtää valmiiseen konttiin kopioimalla tai volumen avulla.



KUVA 1. Kontin ja virtuaalikoneen erot (https://www.docker.com/what-container)

2.3 Node ja Express

Node on asynkrooninen JavaScrtipt ajoympäristö, joka pohjautuu Googlen V8 JavaScript-moottoriin. Vaikka Node on palvelinpuolen ajoympäristö sen pakettienhallintaan (npm) käytetään hyvin laajasti kaikenlaisissa JavaScript projekteissa. Node on saavuttanut suurta suosiota, koska se mahdollistaa palvelin- ja asiakaskoodin kirjoittamisen samalla ohjelmointikielellä.

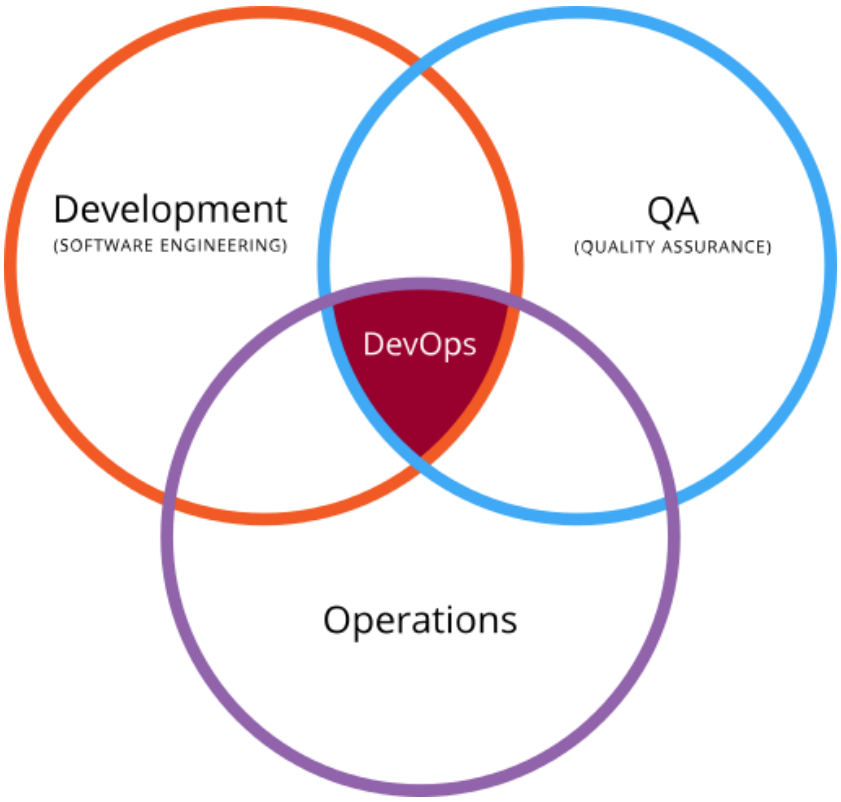
Express on erittäin suosittu Node ohjelmistokehys web-sovellusten kehittämiseen. Express tarjoaa työkaluja muun muassa HTTP-pyyntöjen ja -vastausten käsittelyyn, ja reitittämiseen. Reitittämin tarkoittaa web-ohjelmistoissa HTTP-pyyntöjen ohjaamista oikeille funktioille.

Iso osa kummankin työkalun suosiota on erittäin laaja kolmansien osapuolten kirjastojen tuki. Kirjastot ovat helposti saatavilla npm-pakettivarastosta osoitteessa https://www.npmjs.com/.

3 DEVOPS

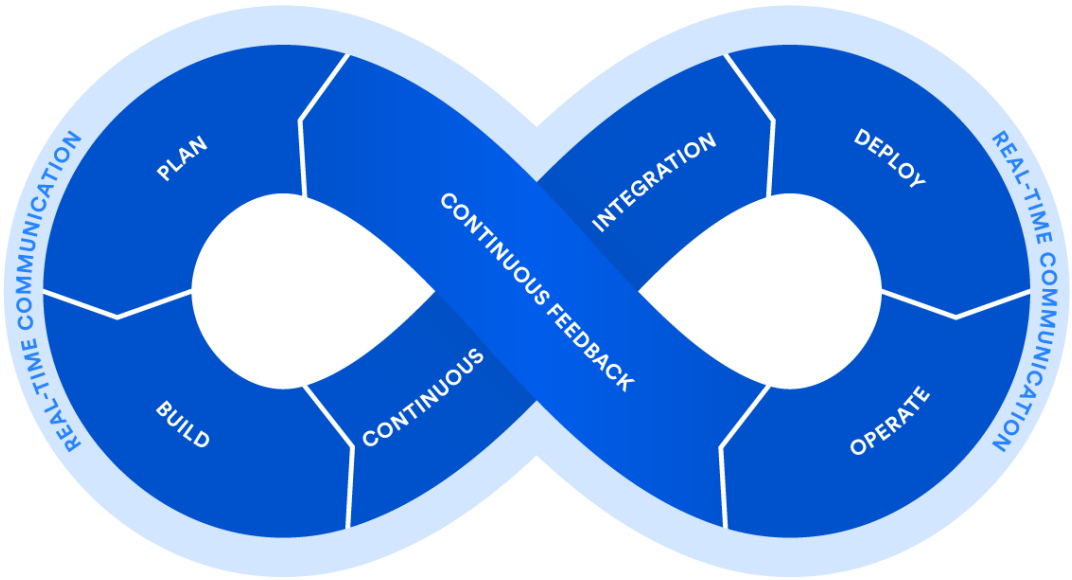
DevOps on sanojen development ja operations sanojen yhdistelmä, joka tarkoittaa ohjelmistotuotannon uudehkoa suuntausta, joka on ikään kuin ketterän kehityksen seuraava kehitys askel. DevOps käytännössä yhdistyy ohjelmistokehitys, -ylläpito ja laadunvalvonta käytännöt runsaan automaation avulla (KUVA 2).

> ” DevOps represents a change in IT culture, focusing on rapid IT service delivery through the adoption of agile, lean practices in the context of a system-oriented approach. DevOps emphasizes people (and culture), and seeks to improve collaboration between operations and development teams. DevOps implementations utilize technology — especially automation tools that can leverage an increasingly programmable and dynamic infrastructure from a life cycle perspective.” (Gartner, DevOps, https://www.gartner.com/it-glossary/devops, Luettu 13.04.2018)



KUVA 2. DevOps Venn diagrammi (https://upload.wikimedia.org/wikipedia/commons/thumb/b/b5/Devops.svg/512px-Devops.svg.png)

DevOps käytäntö pyrkii tehostamaan koko ohjelmistotuotantoprosessia automatisoimalla mahdollisimman monta osaa siitä. Jatkuvuus on erittäin keskeinen käsite DevOps-mailmassa, kaikki prosessit ovat jatkuvia ja inkrementaalisia (KUVA 3).



KUVA 3. DevOps käsitteet (https://www.atlassian.com/devops)

3.1 Automaatio DevOps käytännössä

Automaatio mahdollista jatkuvuuden. Jatkuvan integraation järjestelmä tarkkailee versionhallintaa koodimuutosten varalta ja laukaisee automaattisen käännöksen, käännöksen jälkeen ajetaan kattava määrä automaattitestejä. Jos uusi koodi läpäisee testit, siirtyy järjestelmä automaattiseen julkaisuun. Jos testit epäonnistuvat, järjestelmä huomauttaa kehitystiimiä, jotta vika voidaan korjata. Kun ohjelmisto on julkaistu, sen toimintaa seurataan analysoimalla eri lokitiedostojen koosteita.

3.2 Docker DevOps käytännössä

Docker on kasvattanut suosiotaan DevOpsin kanssa käsi kädessä, sillä se helpottaa automaatiota ja poistaa tuotanto- ja testausympäristöjen konfiguraatioeroista johtuvat ongelmat. Konttien käyttäminen yhdessä mikropalveluarkkitehtuurin kanssa mahdollistaa järjestelmän erittäin nopean skaalautumisen. Kontin käynnistymiseen kuluu korkeintaan se-

kunteja, joten järjestelmää voidaan skaalata reaaliajassa käyttäjien mukaan. Loki koosteista voidaan puolestaan ennustaa käyttäjämääriä, joten kontteja voidaan nostaa valmiiksi ruuhka-aikoina mahdollisimman hyvän käyttäjäkokemuksen takaamiseksi.

3.3 DevOps edut ja haitat

DevOpsin isoin etu on nopeus. Kaikki tapahtuu nopeammin automaation ansiosta: virheet huomataan nopeammin, uudet ominaisuudet saadaan käyttöön nopeammin, järjestelmä skaalautuu nopeammin. Automaatio vähentää myös huomattavasti manuaalista ”oheis”-työtä.

DevOps vaatii järeän automaatiojärjestelmän toimiakseen ja sen rakentaminen on työlästä. Useita työkaluja tulee asentaa ja testitapauksia tulee kirjoittaa valtava määrä. DevOps käytäntöjen käyttöönotto vaatii myös potentiaalisesti suuria organisaatiomuutoksia. Perinteisesti varsinkin suurissa ohjelmistoyrityksissä kehitys ja ylläpitotiimit ovat vahvasti toisistaan erilliset ja kummallakin osastolla on usein paljon vakiintuneita käytäntöjä. Näiden organisaatioiden yhdistäminen samalla kun otetaan paljon uusia työkaluja käyttöön on haastavaa.

4 SUUNNITTELU

Ohjelman toteutuksessa oli tarkoitus hyödyntää mahdollisimman paljon valmiita kirjastoja koodin selkeyttämiseksi ja tehostamiseksi. Esimerkiksi ohjelman pohjana käytettiin Express Node.js sovelluskehystä, joka tarjoaa laajan tuen kolmansien osapuolien kirjastoille ja väliohjelmistoille.

4.1 Vaatimukset

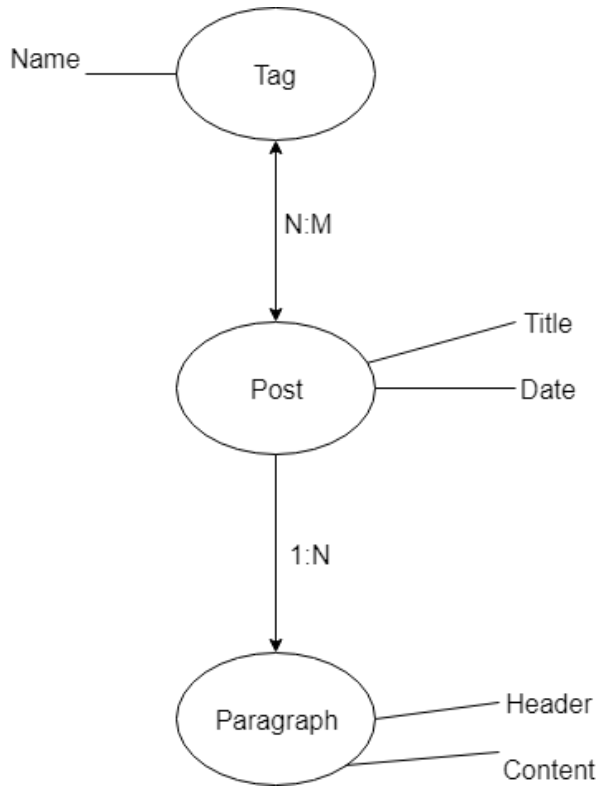
Ohjelmalla oli teknisenä vaatimuksena toteuttaa REST-tyyppinen ohjelmistorajapinta ja toimia Docker-kontissa sekä vaatia käyttäjän todennusta käsiteltäessä tiettyjä resursseja. Ohjelman kontittaminen mahdollistaa vaivattoman julkaisun ja poistaa mahdolliset tuotanto- ja testausympäristöjen eroista johtuvat ongelmat. Koska ohjelmistoa ei ole suunnattu valtaville käyttäjämäärille voidaan järjestelmän kerroksittaisuus jättää huomiotta.

Ohjelman on tarkoitus tukea vain yhtä pääkäyttäjää, joka voi tuottaa ja muokata sisältöä. Käyttäjä tunnistautuu sähköpostin ja salasanan avulla.

Koska ohjelmiston tulee olla tilaton, käyttäjän tunnistamiseen ei voitu käyttää perinteistä sessioihin perustuvaa tunnistautumista. Sessioiden sijaan oli käytettävä käyttöoikeustietueita (Access Token).

4.2 Tietorakenteet

Blogi koostuu postauksista (KUVA 4). Postaukset koostuu edellen kappaleista ja kappaleet otsikoista ja sisällöstä. Postauksilla voi olla myös tageja. Yksi tagi voi liittyä moneen postaukseen. Postauskella on voi olla monta kappaletta, mutta kappaleella liityy aina vain yhteen postaukseen.



KUVA 4. Tietorakenteet

Tageilla on vain yksi ominaisuus; nimi. Postaukseen kuuluu otsikko sekä luonti- ja muokkauspäivämäärä. Kappaleilla puolestaan on oma otsikkonsa ja tekstisisältö.

4.3 Resurssit

Jokaiselle oliolle täytyy olla CRUD (Create, read, update and delete) toiminnot (TAULUKKO 1). Resurssit on suunniteltu HTTP-verbejä silmällä pitäen. GET metodi on tietojen lukemista varten, POST luomista, PATCH päivittämistä ja DELETE poistoa varten. Suojatut reitit vaativat tunnistautumisen.

TAULUKKO 1. käytettävissä olevat resurssit

Method	ULR	Protected				
POST	/paragraphs/	Yes				
PATCH	/paragraphs/id	Yes				
DELETE	/paragraphs/id	Yes				
GET	/paragraphs/post/id	No				
POST	/posts/	Yes				
PATCH	/posts/id	Yes				
DELETE	/posts/id	Yes				
GET	/posts	No				
GET	/posts/eager	No				
POST	/posts/id/tags	Yes				
DELETE	/posts/id/tags	Yes				
POST	/tags/	Yes				
PATCH	/tags/id	Yes				
DELETE	/tags/id	Yes				
GET	/tags	No				
GET	/tags/id/posts	No				
POST	/user/create	No				
PATCH	/users	Yes				
DELETE	/user	Yes				
PATCH	/user/login	No				

5 TOTEUTUS

5.1 Docker

Ohjelma koostuu selkeästi kahdesta osasta: tietokannasta ja web-rajapinnasta. Molemmille osille tarvittiin oma konttinsa. Docker Compose on työkalu, joka mahdollistaa useiden konttien ja niiden välisten suhteiden määrittelyn yhdessä tiedostossa (KUVA 5). Docker Composella voidaan hallita myös kaikkia samassa tiedostossa määriteltyjä konteja samaan aikaan, ne voidaan esimerkiksi käynnistää komennolla ”docker-compose up”.

docker-compose.yml

```yaml
version: '3'
services:
  blog:
    build:
      context: .
      args:
        workdir: ${WORKDIR}
    command: npm run ${RUN_COMMAND}
    volumes:
      - ".:${WORKDIR}"
      - "${WORKDIR}/node_modules"
    ports:
      - "${PORT}:${PORT}"
    networks:
      - backend
    depends_on:
      - db
    environment:
      DATABASE_URL: postgres://${DB_USER}:${DB_PASSWORD}@db/${DB_NAME}
  db:
    image: postgres:10-alpine
    restart: always
    volumes:
      - "./postgres/data:/var/lib/postgresql/data"
    expose:
      - "5432"
    networks:
      - backend
    environment:
      POSTGRES_USER: "${DB_USER}"
      POSTGRES_PASSWORD: "${DB_PASSWORD}"
      POSTGRES_DB: "${DB_NAME}"

networks:
  backend:
    driver: bridge
```

KUVA 5. Compose-tiedosto

Compose-tiedoston käyttäminen on huomattavasti selkeämpää kuin yksittäisten Dockerfilejen (KUVA 6). Dockerfile on Dockerin konttien määrittämiseen käytetty konfiguraatiotiedosto.

Dockerfile

```
FROM node:9

ARG workdir

USER node

USER root
# use changes to package.json to force Docker not to use the cache
# when we change our application's nodejs dependencies:
COPY package.json /tmp/package.json
RUN cd /tmp && npm install --quiet
RUN mkdir -p $workdir && cp -r /tmp/node_modules $workdir && chown -R node:node $workdir

COPY . $workdir

USER node
WORKDIR $workdir
```

KUVA 6. Dockerfile

Compose-tiedostossa määritetään tietokannaksi PostgreSQL käyttäen virallista version 10 Alpine-Linuxiin pohjautuvaa Docker-kuvaa. Web-rajapinta määritetään käyttämään samassa kansiossa olevaa Dockerfileä (KUVA 6). Konttien välille määritetään silta tyyppinen verkko, jolloin tietokantaan ei pääse käsiksi konttien ulkopuolelta lainkaan. Kummallekin kontille annetaan ympäristömuuttujia ”${MUUTTUJAN_NIMI}” syntaksilla. Esimerkiksi tietokannan käyttäjä ja salasana asetetaan ympäristömuuttujilla.

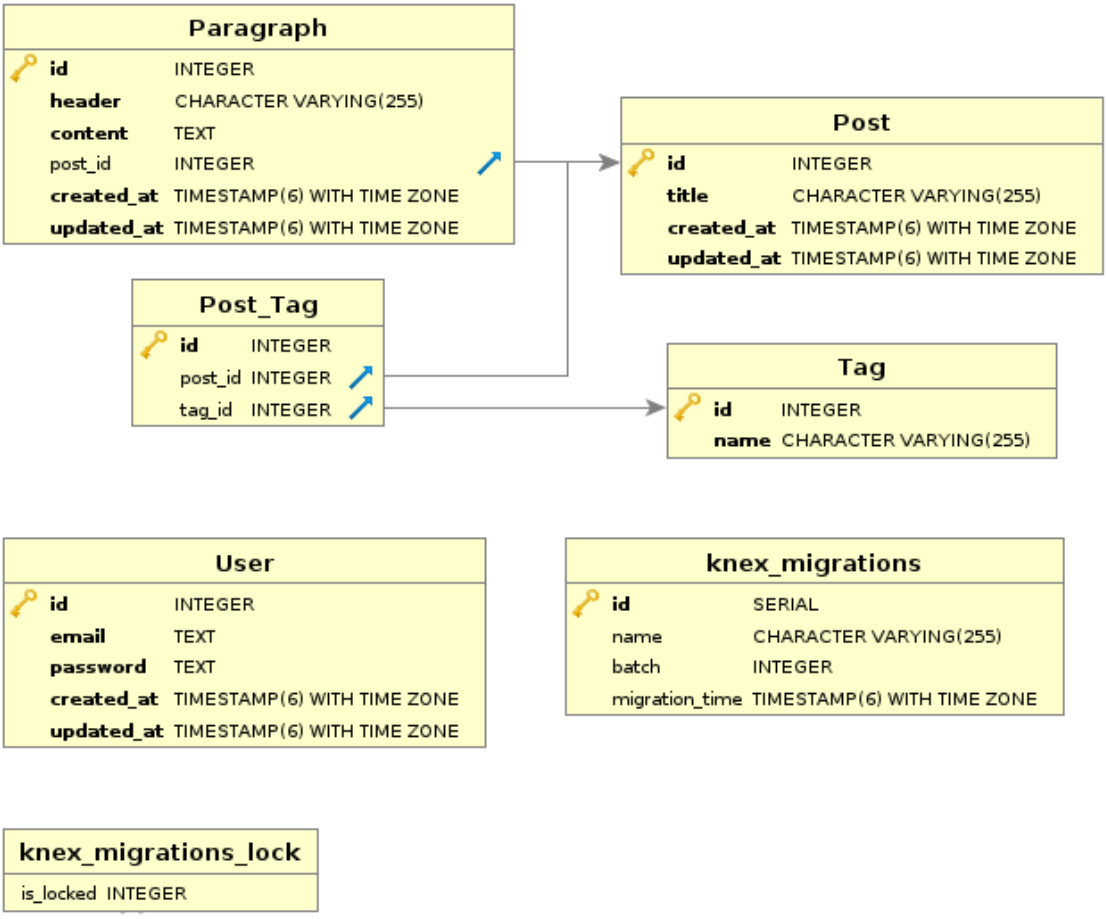
Ympäristömuutujat on määritelty erillisessä ”.env”-tiedostossa (KUVA 7). Ympäristömuuttujien erottaminen erilliseen tiedostoon mahdollistaa niiden jättämisen pois versiohallinasta, jolloin lähdekoodi voidaan julkaista ilman käytettyjä salasanoja.

```
.env
1  DB_USER=Esimerkki
2  DB_PASSWORD=EsimerkkiSalasana
3  DB_NAME=MinumOmaBlogi
4  RUN_COMMAND=dev
5  #RUN_COMMAND=prod
6  WORKDIR=/home/node/app
7  NODE_ENV=production
8  PORT=3000
9  JWT_KEY=SatunnainePitkäMerkkijonotTähän
```

KUVA 7. Ympäristömuutujat

5.2 Tietokanta

PostgreSQL valittiin tietokannaksi luotettavuutensa ja hyvän maineensa ansiosta. Suunniteltujen tietorakenteiden perusteella luotiin tietokantaan taulut (KUVIO 1).



KUVIO 1. ER-kaavio

5.3 Object-relational mapping

Tietokannan käsittelyyn käytetään Objection.js kirjastoa, joka pohjautuu puolestaan Knex.js kirjastoon. Knex helpottaa SQL-kyselyiden kirjoittamista ja Objection mahdollistaa tietojen käsittelyn puhtaasti olio-mallilla.

Knex tarjoaa helpon tavan luoda ja muokata tietokannan tauluja ohjelmallisesti – skeema migraatiot. Migraatiotiedostossa määritetään mitä taululle tehdään, kun migraatio ajetaan tai palautetaan (KUVA 8).

KUVA 8. Kappaletaulun luonti

Objection tarvitsee mallin oliosta, jonka perusteella se osaa yhdistää tietokannan datan olion attribuutteihin. Mallissa määritetään esimerkiksi käytettävän tietokantataulun nimi ja datatyypit (KUVA 9).

Paragraph.js

```
'use strict';

const Model = require('objection').Model;


class Paragraph extends Model {

  static get tableName() {
    return 'Paragraph';
  }

  static get jsonSchema() {
    return {
      type: 'object',
      required: ['header','content','post_id'],

      properties: {
        id: { type: 'integer' },
        post_id: { type: ['integer'] },
        header: { type: 'string', minLength: 1, maxLength: 255 },
        content: { type: 'string', minLength: 1},
      }
    };
  }

  static relationMappings() {
    return{
      post: {
        relation: Model.BelongsToOneRelation,
        modelClass:  __dirname + '/Post',
        join: {
          from: 'Paragraph.post_id',
          to: 'Post.id'
        }
      }
    }
  }
}

module.exports = Paragraph;
```

KUVA 9. Kappaleen Objection malli

5.4 Express

Tämän ohjelmiston tapauksessa väliohjelmistoja hyödynnetään HTTP-pyyntöjen parsimiseen, jotta niitä olisi helpompi käsitellä (KUVA 10).

```
31  //midleware
32  app.use(helmet());
33  app.use(compression());
34  app.use(favicon(path.join(__dirname, 'public/images', 'favicon.ico')));
35  app.use(logger('dev'));
36  app.use(bodyParser.json());
37  app.use(bodyParser.urlencoded({ extended: false }));
38  app.use(cookieParser());
39  app.use(express.static(path.join(__dirname, 'public')));
40
```

KUVA 10. Väliohjelmistojen rekisteröinti

Jokaiselle oliolle on omat reittinsä, joiden kautta niiden resursseihin pääse käsiksi (KUVA 11). Reittimääritykset on jaettu omiin tiedostoihinsa koodin selkeyttämiseksi.

```
55  // Route registration
56  app.use('/', index);
57  app.use('/posts', postRoutes);
58  app.use('/paragraphs', paragraphRoutes);
59  app.use('/tags',tagRoutes);
60  app.use('/user',userRoutes);
```

KUVA 11. Sovelluksen reitit

Esimerkiksi Kappaleen luonti tapahtuu lähettämällä POST – tyyppinen pyyntö osoitteiseen ”/paragraphs” (KUVA 12). Kaikki ”/paragraphs” osoitteeseen tulevat pyynnöt ohjataan pragraphRoutes tiedostolle. Routes tiedostoissa on määritelty kaikki olion tarvitsemat reitit.

```
39  /* CREATE new paragraph */
40  router.post('/',
41      [check('header').exists(),
42      check('content').exists(),
43      check('post_id').exists().isInt({min:1}),
44      sanitize(['header','content'])],
45      checkValidationResult,
46      checkAuth,
47      paragraphController.create);
```

KUVA 12. Kappaleen luonti reitti

Ennen kappaleen luontia tarkastetaan, että vaaditut tiedot on lähetetty, sitten ne puhdistetaan SQL-injektioiden varalta, seuraavaksi tarkastetaan tunnistautumisen oikeellisuus ja lopulta kutsutaan Kappaleen luontifunktiota. Jokaisen olion datan käsittely on eriytetty Controller-tiedostoon koodin selkeyttämiseksi.

Varsinainen Kappaleen luonti tapahtuu siis ”paragraphController.js” tiedostossa (KUVA 13). Tietojen käsittely on varsin yksinkertaista käytettyjen kirjastojen ansiosta.

```
50  //create paragraph
51  exports.create = async (req, res, next) => {
52      const graph = req.body;
53
54      try{
55          const insertedGraph = await transaction(Paragraph.knex(), trx => {
56              return (
57                  Paragraph.query(trx)
58                      .allowInsert('[paragraphs.[header,content]]')
59                      .insertGraph(graph)
60              );
61          });
62
63          res.send(insertedGraph);
64      }catch(err){
65          return res.status(500).send();
66      }
67
68  };
```

KUVA 13. Kappaleen luonti

Myös tunnistautumisen tarkastus on äärimmäisen yksinkertaista kirjastojen ansiosta (KUVA 14).

checkAuth.js

```
const jwt = require('jsonwebtoken');

module.exports = (req, res, next) => {
    try {
        const token = req.headers.authorization.split(" ")[1];
        jwt.verify(token, process.env.JWT_KEY, (err, decoded) => {
            if(err){
                return res.status(401).json({
                    message: 'Auth failed'
                });
            }

            req.userData = decoded;
            next();
        });
    }catch (error) {
        return res.status(401).json({
            message: 'Auth failed'
        });
    }
};
```

KUVA 14. Tunnistautumisen tarkistus

6 YHTEENVETO

6.1 Ongelmat

Ongelmia oli konttien tietojen säilytyksessä. Oletuksena Docker poistaa kontin datan kun kontti sammutetaan, mutta tämän sovelluksen puitteissa tietokannan data ja node_modules kansio oli säilyttävä vaikka kontin sammuttaisi. ”node_modules”- kansiossa on asennettujen kolmansien osapuolien kirjastojen tiedostot. Kirjastojen uudelleen asentamisessa joka kerta, kun kontti käynnistyy, kuluu huomattavan paljon aikaa, joten kansion oli säilyttävä käynnistysten välillä. Kummankin datan säilytyksen ongelmat johtuivat virheellisistä ”volume”-konfiguraatioiden poluista docker-compose tiedostossa.

6.2 Puutteet

Ohjelma ei päässyt sille asetettuihin tavoitteisiin, osin aikataulu rajoitteista johtuen. Testiautomaatio jäi uupumaan täysin, joten ohjelma ei ole kelvollinen esimerkki DevOps tyyppisestä projektista. Minkäänlaista lokitietojen keräystä ei toteutettu. Koodiin jäi myös paljon parannettavan varaa, erityisesti dokumentaation osalta.

6.3 Onnistumiset

Sovellus on kuitenkin kontissa toimiva ja se pystyy toimimaan yksinkertaisen blogin palvelinpuolena. Vaikka koodin dokumentaatio on puutteellinen, rajapinnan käyttö on dokumentoitu kohtuullisen hyvin. Jokaiselle resurssille on esimerkki pyyntö ja vastaus. Projektin hallinta onnistui myös kohtalaisen hyvin GiHubin Projects- työkalun avulla.

6.4 Ohjelman tulevaisuus

Ensin koko ohjelmiston koodi tulee siistiä ja sitten korjata ongelma ja puutteet. Seuraavaksi täytyy kirjoitta yksikkö- ja rajapintatestitapaukset, jotta ohjelmisto olisi oikeasti valmis jatkuvaan integraatioon. Lopuksi uutena toiminnallisuutena toteutetaan kuvien lisääminen, sillä blogi ilman kuvia on melko askeettinen. Aivan viimeisenä voidaan harkita

jonkinlaista Twitter integraatiota, jossa uudet postaukset jaettaisiin automaattisesti Twitterissä tägejä hastageina käyttäen.

LÄHTEET

Michael Wales, 3 Web Dev Careers Decoded: Front-End vs Back-End vs Full Stack, https://blog.udacity.com/2014/12/front-end-vs-back-end-vs-full-stack-web-developers.html, Luettu 08.03.2018

Pluralsight, What's the Difference Between the Front-End and Back-End?, https://www.pluralsight.com/blog/film-games/whats-difference-front-end-back-end, Luettu 08.03.2018

Introducing JSON,https://www.json.org/, Luetu 30.03.2018

Auth0, Introduction to JSON Web Tokens, https://jwt.io/introduction/, Luettu 22.03.2018

e-satis & Jens, What is an ORM and where can I learn more about it?, https://stackoverflow.com/a/1279678, Luettu 30.03.2018

Sakari Hoisko, Kontita koodisi - Dockerize everything - Continuous Delivery with docker in nut shell., https://www.slideshare.net/SakariHoisko/kontita-koodisi-dockerize-everything-continuous-delivery-with-docker-in-nut-shell, Luettu 23.03.2018

Docker, What is Docker, https://www.docker.com/what-docker, Luettu 10.01.2018

John Lees-Miller, Docker Chat Demo, https://github.com/jdleesmiller/docker-chat-demo/blob/master/docker-compose.yml, Luettu 13.03.2018

John Lees-Miller, Docker Chat Demo, https://github.com/jdleesmiller/docker-chat-demo/blob/master/Dockerfile, Luettu 13.03.2018

Tierney Cyren, 8 Protips to Start Killing It When Dockerizing Node.js, https://nodesource.com/blog/8-protips-to-start-killing-it-when-dockerizing-node-js/ , Luettu 13.03.2018

WWW3, Token Based Authentication -- Implementation Demonstration, https://www.w3.org/2001/sw/Europe/events/foaf-galway/papers/fp/token_based_authentication/, Luettu 16.03.2018

Express, Routing, https://expressjs.com/en/guide/routing.html , Luettu 14.02.2018

Express, Using middleware, https://expressjs.com/en/guide/using-middleware.htm, Luettu 23.02.2018l

Docker, Docker Compose, https://docs.docker.com/compose/, Luettu 18.01.2018

Knex.js, http://knexjs.org/, Luettu 12.02.2018

Vincit, Objection.js, https://vincit.github.io/objection.js/, Luettu 12.02.2018

Amazon Web Services, What is DevOps?, https://aws.amazon.com/devops/what-is-devops/, Luettu 13.04.2018

Amazon Web Services, What is Continuous Delivery?, https://aws.amazon.com/devops/continuous-delivery/, Luettu 13.04.2018

Puppet, DevOps, https://puppet.com/solutions/devops, Luettu 13.04.2018

Gartner, DevOps, https://www.gartner.com/it-glossary/devops, Luettu 13.04.2018

Microsoft, DevOps and Microsoft, https://www.visualstudio.com/devops/, Luettu 13.04.2018

Atlassian, DevOps: Breaking the Development-Operations barrier, https://www.atlassian.com/devops, Luettu 13.04.2018

David Gonzalez, Developing Microservices with Node.js, Packt Publishing 2016. ISBN 9781785887406

Caio Pereira Ribeiro, Building APIs with Node.js, Apress 2016. ISBN 9781484224427

Sakari Hoisko, Tampere Docker meetup - Happy 5th Birthday Docker, https://www.slideshare.net/SakariHoisko/tampere-docker-meeup-happy-5th-birthday-docker, Luettu 22.03.2018

Docker, What is a container, https://www.docker.com/what-container, Luettu 15.04.2018

RESTfulAPI.net, REST Architectural Constraints, https://restfulapi.net/rest-architectural-constraints/, Luettu 15.04.2018